**import** \* **as** firebase **from** 'firebase/app';  
**import** { Injectable } **from** '@angular/core';  
**import** {Router} **from** '@angular/router';  
**import** {AngularFireAuth} **from** 'angularfire2/auth';  
**import** {Observable} **from** 'rxjs';  
  
@Injectable({  
 providedIn: 'root'  
})  
**export class** AuthService {  
 **public** user$: Observable<firebase.User>;  
  
 **constructor**(**private** router: Router, **private** afAuth: AngularFireAuth) {  
 **this**.user$ = **this**.afAuth.authState;  
 }  
  
 login(email, password) {  
 **this**.afAuth.auth.signInWithEmailAndPassword(email, password)  
 .then(\_ => **this**.router.navigate(['/modeldb']))  
 .catch(error => console.log('auth error', error));  
 }  
  
 logout() {  
 **this**.afAuth.auth.signOut()  
 .then(\_ => **this**.router.navigate(['/']));  
 }  
}

**import** \* **as** firebase **from** 'firebase/app';  
**import** { Injectable } **from** '@angular/core';  
**import** {Router} **from** '@angular/router';  
**import** {AngularFireAuth} **from** 'angularfire2/auth';  
**import** {Observable} **from** 'rxjs';  
  
@Injectable({  
 providedIn: 'root'  
})  
**export class** AuthService {  
 **public** user$: Observable<firebase.User>;  
  
 **constructor**(**private** router: Router, **private** afAuth: AngularFireAuth) {  
 **this**.user$ = **this**.afAuth.authState;  
 }  
  
 login(email, password) {  
 **this**.afAuth.auth.signInWithEmailAndPassword(email, password)  
 .then(\_ => **this**.router.navigate(['/modeldb']))  
 .catch(error => console.log('auth error', error));  
 }  
  
 logout() {  
 **this**.afAuth.auth.signOut()  
 .then(\_ => **this**.router.navigate(['/']));  
 }  
}

#!/usr/bin/env node  
  
*/\*\*  
 \* Module dependencies.  
 \*/***var** app = require('../app');  
**var** debug = require('debug')('ModelKBDB:server');  
**var** http = require('http');  
  
*/\*\*  
 \* Get port from environment and store in Express.  
 \*/***var** port = normalizePort(process.env.PORT || '3000');  
app.set('port', port);  
  
*/\*\*  
 \* Create HTTP server.  
 \*/***var** server = http.createServer(app);  
  
*/\*\*  
 \* Listen on provided port, on all network interfaces.  
 \*/*server.listen(port);  
server.on('error', onError);  
server.on('listening', onListening);  
  
*/\*\*  
 \* Normalize a port into a number, string, or false.  
 \*/***function** normalizePort(val) {  
 **var** port = parseInt(val, 10);  
  
 **if** (isNaN(port)) {  
 // named pipe  
 **return** val;  
 }  
  
 **if** (port >= 0) {  
 // port number  
 **return** port;  
 }  
  
 **return false**;  
}  
  
*/\*\*  
 \* Event listener for HTTP server "error" event.  
 \*/***function** onError(error) {  
 **if** (error.syscall !== 'listen') {  
 **throw** error;  
 }  
  
 **var** bind = **typeof** port === 'string'  
 ? 'Pipe ' + port  
 : 'Port ' + port;  
  
 // handle specific listen errors with friendly messages  
 **switch** (error.code) {  
 **case** 'EACCES':  
 console.error(bind + ' requires elevated privileges');  
 process.exit(1);  
 **break**;  
 **case** 'EADDRINUSE':  
 console.error(bind + ' is already in use');  
 process.exit(1);  
 **break**;  
 **default**:  
 **throw** error;  
 }  
}  
  
*/\*\*  
 \* Event listener for HTTP server "listening" event.  
 \*/***function** onListening() {  
 **var** addr = server.address();  
 **var** bind = **typeof** addr === 'string'  
 ? 'pipe ' + addr  
 : 'port ' + addr.port;  
 debug('Listening on ' + bind);  
}

**var** mongoose = require('mongoose');  
  
**var** ExperimentSchema = **new** mongoose.Schema({  
 model\_name: String,  
 framework: String,  
 size: String,  
 epochs: String,  
 layersCount: String,  
 InputTensors: String,  
 OutputTensor: String,  
 Optimizer: String,  
 LossFunction: String,  
 AccuracyValue: String,  
 LossValue: String,  
 },  
 {collection:'Experiment'}  
);  
  
**const** Experiment = mongoose.model('Experiment',ExperimentSchema);  
module.exports = Experiment;

**var** express = require('express');  
**var** router = express.Router();  
**var** mongoose = require('mongoose');  
**var** Experiment = require('../models/experiment.js');  
  
/\* GET ALL EXPERIMENTS \*/  
router.get('/', **function**(req, res, next) {  
 Experiment.find(**function** (err, products) {  
 **if** (err) **return** next(err);  
 res.json(products);  
 });  
});  
  
/\* GET SINGLE EXPERIMENT BY ID \*/  
router.get('/:id', **function**(req, res, next) {  
 Experiment.findById(req.params.id, **function** (err, post) {  
 **if** (err) **return** next(err);  
 res.json(post);  
 });  
});  
  
/\* SAVE EXPERIMENT \*/  
router.post('/', **function**(req, res, next) {  
 Experiment.create(req.body, **function** (err, post) {  
 **if** (err) **return** next(err);  
 res.json(post);  
 });  
});  
  
/\* UPDATE EXPERIMENT \*/  
router.put('/:id', **function**(req, res, next) {  
 Experiment.findByIdAndUpdate(req.params.id, req.body, **function** (err, post) {  
 **if** (err) **return** next(err);  
 res.json(post);  
 });  
});  
  
/\* DELETE EXPERIMENT \*/  
router.delete('/:id', **function**(req, res, next) {  
 Experiment.findByIdAndRemove(req.params.id, req.body, **function** (err, post) {  
 **if** (err) **return** next(err);  
 res.json(post);  
 });  
});  
  
module.exports = router;

<nav class="navbar">  
 <a class="navbar-left" href="#">  
 <img src="../../assets/kb\_logo.png" width="150" height="150" class="d-inline-block align-top" alt="">  
 <a class="navbar-text" href="#">  
 <h1>ModelKB DB</h1>  
 </a>  
 <a class="navbar-right" href="#" *\*ngIf*="(authService.user$ | async)?.uid">  
 <button class="btn btn-light" (click)="authService.logout()">Logout</button>  
 </a>  
 </a>  
</nav>

<form #LoginForm="ngForm" (ngSubmit)="loginEvent(LoginForm.value)">  
 <div class="panel primary-panel">  
 <div class="panel-heading">  
 <h2 class="panel-title">Login</h2>  
 </div>  
 <div class="panel-body">  
 <div class="form-group">  
 <label>Email Address:</label>  
 <input type="text" required class="form-control" name="email" ngModel>  
 </div>  
 </div>  
 </div>  
 <div class="panel primary-panel">  
 <div class="form-group">  
 <label>Password:</label>  
 <input required type="password" class="form-control" name="password" ngModel>  
 </div>  
 </div>  
  
 <div class="panel primary-panel">  
 <button class="btn btn-primary">Login</button><br><br>  
 <h6>New User? <a routerLink="/register">Register</a></h6>  
 <h6>Forgot your password? <a routerLink="/register">Reset</a></h6>  
 </div>  
</form>

**import** { Component, OnInit } **from** '@angular/core';  
**import** { AuthService } **from** '../auth/auth.service';  
  
@Component({  
 selector: 'app-login',  
 templateUrl: './login.component.html',  
 styleUrls: ['./login.component.css']  
})  
**export class** LoginComponent **implements** OnInit {  
  
 **private** email: **string**;  
 **private** password: **string**;  
 alert = '';  
  
 **constructor**(**public** authService: AuthService) {}  
  
 loginEvent(value: **any**) {  
 **this**.email = value.email.toString();  
 **this**.password = value.password.toString();  
 console.log(**this**.email);  
 console.log(**this**.password);  
  
 **if** (**this**.email !== '' && **this**.password !== '') {  
 **this**.authService.login(**this**.email, **this**.password);  
 }  
 }  
  
 ngOnInit() {  
 }  
  
}

<div class="button-row">  
 <a mat-raised-button color="secondary" [routerLink]="['/modeldb/upload']"><mat-icon>add</mat-icon></a>  
</div>  
<div class="example-container mat-elevation-z8">  
 <table mat-table #table [dataSource]="dataSource">  
  
 <!--- Note that these columns can be defined in any order.  
 The actual rendered columns are set as a property on the row definition" -->  
  
 <!-- model\_name Column -->  
 <ng-container matColumnDef="model\_name">  
 <th mat-header-cell *\*matHeaderCellDef*> model\_name </th>  
 <td mat-cell *\*matCellDef*="**let** element" class="model\_name-col"> {{element.model\_name}} </td>  
 </ng-container>  
  
 <!-- InputTensors Column -->  
 <ng-container matColumnDef="InputTensors">  
 <th mat-header-cell *\*matHeaderCellDef*> InputTensors </th>  
 <td mat-cell *\*matCellDef*="**let** element"> {{element.InputTensors}} </td>  
 </ng-container>  
  
 <!-- OutputTensor Column -->  
 <ng-container matColumnDef="OutputTensor">  
 <th mat-header-cell *\*matHeaderCellDef*> OutputTensor </th>  
 <td mat-cell *\*matCellDef*="**let** element"> {{element.OutputTensor}} </td>  
 </ng-container>  
  
 <!-- Optimizer Column -->  
 <ng-container matColumnDef="Optimizer">  
 <th mat-header-cell *\*matHeaderCellDef*> Optimizer </th>  
 <td mat-cell *\*matCellDef*="**let** element"> {{element.Optimizer}} </td>  
 </ng-container>  
  
 <!-- AccuracyValue Column -->  
 <ng-container matColumnDef="AccuracyValue">  
 <th mat-header-cell *\*matHeaderCellDef*> AccuracyValue </th>  
 <td mat-cell *\*matCellDef*="**let** element"> {{element.AccuracyValue}} </td>  
 </ng-container>  
  
 <tr mat-header-row *\*matHeaderRowDef*="displayedColumns"></tr>  
 <tr mat-row *\*matRowDef*="**let** row; columns: displayedColumns;" [routerLink]="['/modeldb/details/', row.\_id]"></tr>  
 </table>  
</div>

**import** { Component, OnInit } **from** '@angular/core';  
**import** { ApiService } **from** '../api.service';  
**import** { DataSource } **from** '@angular/cdk/collections';  
**import** { Observable } **from** 'rxjs';  
  
@Component({  
 selector: 'app-modeldb',  
 templateUrl: './modeldb.component.html',  
 styleUrls: ['./modeldb.component.css']  
})  
**export class** ModeldbComponent **implements** OnInit {  
  
 experiments: **any**;  
 displayedColumns = ['model\_name', 'InputTensors', 'OutputTensor', 'Optimizer', 'AccuracyValue'];  
 dataSource = **new** ExperimentDataSource(**this**.api);  
  
 **constructor**(**private** api: ApiService) { }  
  
 ngOnInit() {  
 **this**.api.getExperiments()  
 .subscribe(res => {  
 console.log(res);  
 **this**.experiments = res;  
 }, err => {  
 console.log(err);  
 });  
 }  
  
}  
  
**export class** ExperimentDataSource **extends** DataSource<**any**> {  
 **constructor**(**private** api: ApiService) {  
 **super**();  
 }  
  
 connect() {  
 **return this**.api.getExperiments();  
 }  
  
 disconnect() {  
  
 }  
}

<div class="button-row">  
 <a mat-raised-button color="primary" [routerLink]="['/modeldb']"><mat-icon>list</mat-icon></a>  
</div>  
<mat-card class="example-card">  
 <mat-card-header>  
 <mat-card-title><h2>{{experiment.model\_name}}</h2></mat-card-title>  
 <mat-card-subtitle>{{experiment.framework}}</mat-card-subtitle>  
 </mat-card-header>  
 <mat-card-content>  
 <dl>  
 <dt>Model Name:</dt>  
 <dd>{{experiment.model\_name}}</dd>  
 <dt>Framework:</dt>  
 <dd>{{experiment.framework}}</dd>  
 <dt>Size:</dt>  
 <dd>{{experiment.size}}</dd>  
 <dt>Epochs</dt>  
 <dd>{{experiment.epochs}}</dd>  
 <dt>Layers Count:</dt>  
 <dd>{{experiment.layersCount}}</dd>  
 <dt>Input Tensors:</dt>  
 <dd>{{experiment.InputTensors}}</dd>  
 <dt>Output Tensor:</dt>  
 <dd>{{experiment.OutputTensor}}</dd>  
 <dt>Optimizer:</dt>  
 <dd>{{experiment.Optimizer}}</dd>  
 <dt>Loss Function:</dt>  
 <dd>{{experiment.LossFunction}}</dd>  
 <dt>Accuracy Value:</dt>  
 <dd>{{experiment.AccuracyValue}}</dd>  
 <dt>Loss Value:</dt>  
 <dd>{{experiment.LossValue}}</dd>  
 </dl>  
 </mat-card-content>  
</mat-card>

**import** { Component, OnInit } **from** '@angular/core';  
**import** { ActivatedRoute, Router } **from** '@angular/router';  
**import** { ApiService } **from** '../api.service';  
  
@Component({  
 selector: 'app-modeldb-detail',  
 templateUrl: './modeldb-detail.component.html',  
 styleUrls: ['./modeldb-detail.component.css']  
})  
**export class** ModeldbDetailComponent **implements** OnInit {  
  
 experiment = {};  
 **constructor**(**private** route: ActivatedRoute, **private** api: ApiService, **private** router: Router) { }  
  
 ngOnInit() {  
 **this**.getExperimentDetails(**this**.route.snapshot.params['id']);  
 }  
  
 getExperimentDetails(id) {  
 **this**.api.getExperiment(id)  
 .subscribe(data => {  
 console.log(data);  
 **this**.experiment = data;  
 });  
 }  
  
}

<body>  
  
<button type="button" mat-raised-button (click)="fileInput.click()">Choose File</button>  
<input hidden (change)="onFileSelected()" #fileInput type="file" id="file">  
  
<div class="button-row">  
 <a mat-raised-button color="primary" [routerLink]="['/modeldb']"><mat-icon>list</mat-icon></a>  
</div>  
<form [formGroup]="experimentForm" (ngSubmit)="onFormSubmit(experimentForm.value)">  
 <mat-form-field appearance="legacy">  
 <input matInput placeholder="model\_name" formControlName="model\_name"  
 [errorStateMatcher]="matcher">  
 <mat-error>  
 <span *\*ngIf*="!experimentForm.get('model\_name').valid && experimentForm.get('model\_name').touched">Please enter model\_name</span>  
 </mat-error>  
 </mat-form-field>  
 <mat-form-field appearance="legacy">  
 <input matInput placeholder="framework" formControlName="framework"  
 [errorStateMatcher]="matcher">  
 <mat-error>  
 <span *\*ngIf*="!experimentForm.get('framework').valid && experimentForm.get('framework').touched">Please enter framework</span>  
 </mat-error>  
 </mat-form-field>  
 <mat-form-field appearance="legacy">  
 <input matInput placeholder="size" formControlName="size"  
 [errorStateMatcher]="matcher">  
 <mat-error>  
 <span *\*ngIf*="!experimentForm.get('size').valid && experimentForm.get('size').touched">Please enter size</span>  
 </mat-error>  
 </mat-form-field>  
 <mat-form-field appearance="legacy">  
 <input matInput placeholder="epochs" formControlName="epochs"  
 [errorStateMatcher]="matcher">  
 <mat-error>  
 <span *\*ngIf*="!experimentForm.get('epochs').valid && experimentForm.get('epochs').touched">Please enter epochs</span>  
 </mat-error>  
 </mat-form-field>  
 <mat-form-field appearance="legacy">  
 <input matInput placeholder="layersCount" formControlName="layersCount"  
 [errorStateMatcher]="layersCount">  
 <mat-error>  
 <span *\*ngIf*="!experimentForm.get('layersCount').valid && experimentForm.get('layersCount').touched">Please enter layersCount</span>  
 </mat-error>  
 </mat-form-field>  
 <mat-form-field appearance="legacy">  
 <input matInput placeholder="InputTensors" formControlName="InputTensors"  
 [errorStateMatcher]="matcher">  
 <mat-error>  
 <span *\*ngIf*="!experimentForm.get('InputTensors').valid && experimentForm.get('InputTensors').touched">Please enter InputTensors</span>  
 </mat-error>  
 </mat-form-field>  
 <mat-form-field appearance="legacy">  
 <input matInput placeholder="OutputTensor" formControlName="OutputTensor"  
 [errorStateMatcher]="matcher">  
 <mat-error>  
 <span *\*ngIf*="!experimentForm.get('OutputTensor').valid && experimentForm.get('OutputTensor').touched">Please enter OutputTensor</span>  
 </mat-error>  
 </mat-form-field>  
 <mat-form-field appearance="legacy">  
 <input matInput placeholder="Optimizer" formControlName="Optimizer"  
 [errorStateMatcher]="matcher">  
 <mat-error>  
 <span *\*ngIf*="!experimentForm.get('Optimizer').valid && experimentForm.get('Optimizer').touched">Please enter Optimizer</span>  
 </mat-error>  
 </mat-form-field>  
 <mat-form-field appearance="legacy">  
 <input matInput placeholder="LossFunction" formControlName="LossFunction"  
 [errorStateMatcher]="matcher">  
 <mat-error>  
 <span *\*ngIf*="!experimentForm.get('LossFunction').valid && experimentForm.get('LossFunction').touched">Please enter LossFunction</span>  
 </mat-error>  
 </mat-form-field>  
 <mat-form-field appearance="legacy">  
 <input matInput placeholder="AccuracyValue" formControlName="AccuracyValue"  
 [errorStateMatcher]="matcher">  
 <mat-error>  
 <span *\*ngIf*="!experimentForm.get('AccuracyValue').valid && experimentForm.get('AccuracyValue').touched">Please enter AccuracyValue</span>  
 </mat-error>  
 </mat-form-field>  
 <mat-form-field appearance="legacy">  
 <input matInput placeholder="LossValue" formControlName="LossValue"  
 [errorStateMatcher]="matcher">  
 <mat-error>  
 <span *\*ngIf*="!experimentForm.get('LossValue').valid && experimentForm.get('LossValue').touched">Please enter LossValue</span>  
 </mat-error>  
 </mat-form-field>  
 <div class="button-row">  
 <button type="submit" [disabled]="!experimentForm.valid" mat-raised-button color="primary"><mat-icon>save</mat-icon></button>  
 </div>  
</form>  
</body>

**import** { Component, OnInit } **from** '@angular/core';  
**import** { Router } **from** '@angular/router';  
**import** { ApiService } **from** '../api.service';  
**import** { FormControl, FormGroupDirective, FormBuilder, FormGroup, NgForm, Validators } **from** '@angular/forms';  
  
@Component({  
 selector: 'app-modeldb-upload',  
 templateUrl: './modeldb-upload.component.html',  
 styleUrls: ['./modeldb-upload.component.css']  
})  
**export class** ModeldbUploadComponent **implements** OnInit {  
  
 experimentForm: FormGroup;  
 model\_name: **string** = '';  
 framework: **string** = '';  
 size: **string** = '';  
 epochs: **string** = '';  
 layersCount: **string** = '';  
 InputTensors: **string** = '';  
 OutputTensor: **string** = '';  
 Optimizer: **string** = '';  
 LossFunction: **string** = '';  
 AccuracyValue: **string** = '';  
 LossValue: **string** = '';  
  
 **constructor**(**private** router: Router, **private** api: ApiService, **private** formBuilder: FormBuilder) { }  
  
 ngOnInit() {  
 **this**.experimentForm = **this**.formBuilder.group({  
 'model\_name' : [**null**, Validators.*required*],  
 'framework' : [**null**, Validators.*required*],  
 'size' : [**null**, Validators.*required*],  
 'epochs' : [**null**, Validators.*required*],  
 'layersCount' : [**null**, Validators.*required*],  
 'InputTensors' : [**null**, Validators.*required*],  
 'OutputTensor' : [**null**, Validators.*required*],  
 'Optimizer' : [**null**, Validators.*required*],  
 'LossFunction' : [**null**, Validators.*required*],  
 'AccuracyValue' : [**null**, Validators.*required*],  
 'LossValue' : [**null**, Validators.*required*]  
 });  
 }  
  
 onFormSubmit(form:NgForm) {  
 **this**.api.postExperiment(form)  
 .subscribe(res => {  
 **let** id = res['\_id'];  
 **this**.router.navigate(['/modeldb/details', id]);  
 }, (err) => {  
 console.log(err);  
 });  
 }  
  
 onFileSelected() {  
 console.log('placeholder');  
 }  
}

<form #RegForm="ngForm" (ngSubmit)="registerEvent(RegForm.value)">  
 <div class="panel primary-panel">  
 <div class="panel-heading">  
 <h2 class="panel-title">New User Registration</h2>  
 </div>  
 <div class="panel-body">  
 <div class="form-group">  
 <label>Email Address:</label>  
 <input type="text" required class="form-control" name="email" ngModel>  
 </div>  
 </div>  
 <div class="panel primary-panel">  
 <div class="form-group">  
 <label>Password:</label>  
 <input required type="password" class="form-control" name="password" ngModel>  
 </div>  
 </div>  
  
 <div class="panel primary-panel">  
 <div class="form-group">  
 <label>Confirm Password:</label>  
 <input required type="password" class="form-control" name="Cpassword" ngModel>  
 </div>  
 </div>  
  
 <div class="panel primary-panel">  
 <button class="btn btn-primary">Register</button><br><br>  
 <h6>Existing User? <a routerLink="/login">Login</a></h6>  
 </div>  
 </div>  
</form>  
  
<div *\*ngFor*="**let** login of logins">{{ login }}</div>

**import** { Component, OnInit } **from** '@angular/core';  
**import** {AngularFireAuth} **from** 'angularfire2/auth';  
  
@Component({  
 selector: 'app-register',  
 templateUrl: './register.component.html',  
 styleUrls: ['./register.component.css']  
})  
**export class** RegisterComponent **implements** OnInit {  
  
 **private** email: **string**;  
 **private** password: **string**;  
 alert = '';  
  
 **constructor**(**private** firebaseauth: AngularFireAuth) { }  
  
 ngOnInit() {  
 }  
  
 registerEvent(value: **any**) {  
 **this**.email = value.email.toString();  
 **this**.password = value.password.toString();  
 console.log(**this**.email);  
 console.log(**this**.password);  
  
 **try** {  
 **this**.firebaseauth.auth.createUserWithEmailAndPassword(**this**.email, **this**.password).then(() => {  
 }).catch(() => {  
 **this**.alert = 'Invalid email/password should be of 6 characters';  
 });  
 } **catch** (e) {  
 console.error(e);  
 }  
  
 }  
  
}

**import** { Injectable } **from** '@angular/core';  
**import** { Observable, of, throwError } **from** 'rxjs';  
**import** { HttpClient, HttpHeaders, HttpErrorResponse } **from** '@angular/common/http';  
**import** { catchError, tap, map } **from** 'rxjs/operators';  
  
**const** httpOptions = {  
 headers: **new** HttpHeaders({'Content-Type': 'application/json'})  
};  
**const** apiUrl = "/api";  
  
@Injectable({  
 providedIn: 'root'  
})  
**export class** ApiService {  
  
 constructor(**private** http: HttpClient) { }  
  
 **private** handleError(error: HttpErrorResponse) {  
 **if** (error.error **instanceof** ErrorEvent) {  
 // A client-side or network error occurred. Handle it accordingly.  
 console.error('An error occurred:', error.error.message);  
 } **else** {  
 // The backend returned an unsuccessful response code.  
 // The response body may contain clues as to what went wrong,  
 console.error(  
 `Backend returned code ${error.status}, ` +  
 `body was: ${error.error}`);  
 }  
 // return an observable with a user-facing error message  
 **return** throwError('Something bad happened; please try again later.');  
 }  
  
 **private** extractData(res: Response) {  
 **let** body = res;  
 **return** body || { };  
 }  
  
 getExperiments(): Observable<**any**> {  
 **return this**.http.get(apiUrl, httpOptions).pipe(  
 map(**this**.extractData),  
 catchError(**this**.handleError));  
 }  
  
 getExperiment(id: **string**): Observable<**any**> {  
 **const** url = `${apiUrl}/${id}`;  
 **return this**.http.get(url, httpOptions).pipe(  
 map(**this**.extractData),  
 catchError(**this**.handleError));  
 }  
  
 postExperiment(data): Observable<**any**> {  
 **return this**.http.post(apiUrl, data, httpOptions)  
 .pipe(  
 catchError(**this**.handleError)  
 );  
 }  
  
 updateExperiment(data): Observable<**any**> {  
 **return this**.http.put(apiUrl, data, httpOptions)  
 .pipe(  
 catchError(**this**.handleError)  
 );  
 }  
  
 deleteExperiment(id: **string**): Observable<{}> {  
 **const** url = `${apiUrl}/${id}`;  
 **return this**.http.delete(url, httpOptions)  
 .pipe(  
 catchError(**this**.handleError)  
 );  
 }  
  
}